**Spring**

**Spring是一个轻量级的控制反转(IoC)和面向切面(AOP)的容器（框架）。**

## Spring配置

**alias 设置别名 , 为bean设置别名 , 可以设置多个别名**

**<!--设置别名：在获取Bean的时候可以使用别名获取-->**

**<alias name="userT" alias="userNew"/>**

**Bean的配置**

<!--**bean就是java对象,由Spring创建和管理**-->

<!--

**id 是bean的标识符,要唯一,**如果**没有配置id,name就是默认标识符**

**如果配置id,又配置了name,那么name是别名**

name可以设置**多个**别名,可以用**逗号,分号,空格**隔开

**如果不配置id和name,可以根据applicationContext.getBean(.class)获取对象**;

**class是bean的全限定名=包名+类名**

-->

<bean id="hello" name="**hello2 h2,h3;h4**" class="com.kuang.pojo.Hello">

<property name="name" value="Spring"/>

</bean>

**import**

**团队的合作通过import来实现**

**<import resource="{path}/beans.xml"/>**

## DI(依赖注入)

**DI(依赖注入)是实现IoC的一种方法，IoC是Spring框架的核心内容**

**控制反转是一种通过描述（XML或注解）并通过第三方去生产或获取特定对象的方式。在Spring中实现控制反转的是IoC容器，其实现方法是依赖注入**

* **依赖 : 指Bean对象的创建依赖于容器 . Bean对象的依赖资源 .**
* **注入 : 指Bean对象所依赖的资源 , 由容器来设置和装配 .**

### 注入方式

**构造器注入 Set 注入**

**扩展的注入： 常量注入**

**<bean id="student" class="com.kuang.pojo.Student">**

**<property name="name" value="小明"/>**

**</bean>**

测试：

@Test

public void test01(){

ApplicationContext

context=new ClassPathXmlApplicationContext("applicationContext.xml");

Student student = (Student) context.getBean("**studen**t");

System.out.println(**student.getName()**);

**Bean注入**

**注意点：这里的值是一个引用，ref**

**<bean id="addr" class="com.kuang.pojo.Address">**

**<property name="address" value="重庆"/>**

**</bean>**

**<bean id="student" class="com.kuang.pojo.Student">**

**<property name="name" value="小明"/>**

**<property name="address" ref="addr"/>**

**</bean>**

**数组注入**

**<bean id="student" class="com.kuang.pojo.Student">**

**<property name="name" value="小明"/>**

**<property name="address" ref="addr"/>**

**<property name="books">**

**<array>**

**<value>西游记</value>**

**<value>红楼梦</value>**

**<value>水浒传</value>**

**</array>**

**</property>**

**</bean>**

**List注入**

**<property name="hobbys">**

**<list>**

**<value>听歌</value>**

**<value>看电影</value>**

**<value>爬山</value>**

**</list>**

**</property>**

**Map注入**

**<property name="card">**

**<map>**

**<entry key="中国邮政" value="456456456465456"/>**

**<entry key="建设" value="1456682255511"/>**

**</map>**

**</property>**

**set注入**

**<property name="games">**

**<set>**

**<value>LOL</value>**

**<value>BOB</value>**

**<value>COC</value>**

**</set>**

**</property>**

**Null注入**

**<property name="wife"><null/></property>**

**Properties注入**

**<property name="info">**

**<props>**

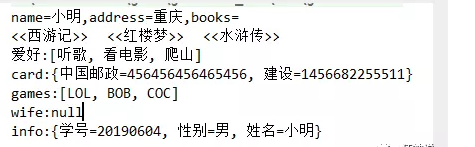
**<prop key="学号">20190604</prop>**

**<prop key="性别">男</prop>**

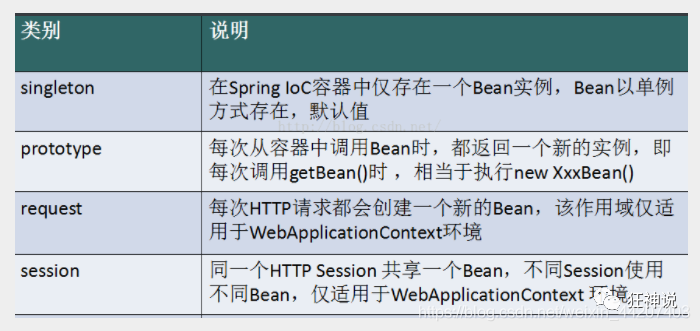
**<prop key="姓名">小明</prop>**

**</props>**

**</property>**



**Bean的作用域**



**在Spring中，那些组成应用程序的主体及由Spring IoC容器所管理的对象，被称之为bean。简单地讲，bean就是由IoC容器初始化、装配及管理的对象**

**Singleton(单例模式)**

当一个bean的作用域为**Singleton**，那么Spring IoC容器中**只会存在一个共享的bean实例**，并且**所有对bean的请求，只要id与该bean定义相匹配，则只会返回bean的同一实例**。Singleton是单例类型，就是在创建起容器时就同时自动创建了一个bean的对象，不管你是否使用，他都存在了，每次获取到的对象都是同一个对象。注意，Singleton作用域是Spring中的缺省作用域。要在XML中将bean定义成singleton，可以这样配置：

**<bean id="ServiceImpl" class="cn.csdn.service.ServiceImpl" scope="singleton">**

**测试：**

**@Test**

**public void test03(){**

**ApplicationContext**

**context = new ClassPathXmlApplicationContext("applicationContext.xml");**

**User user = (User) context.getBean("user");**

**User user2 = (User) context.getBean("user");**

**System.out.println(user==user2);**

**}**

**Prototype(原型模式)**

当一个bean的作用域为**Prototype**，表示**一个bean定义对应多个对象实例**。Prototype作用域的bean会导致在每次对该bean请求（将其注入到另一个bean中，或者以程序的方式调用容器的getBean()方法）时都会创建一个新的bean实例。**Prototype是原型类型，它在我们创建容器的时候并没有实例化，而是当我们获取bean的时候才会去创建一个对象**，**而且我们每次获取到的对象都不是同一个对象**。根据经验，对有状态的bean应该使用prototype作用域，而对无状态的bean则应该使用singleton作用域。在XML中将bean定义成prototype，可以这样配置：

**<bean id="account" class="com.foo.DefaultAccount" scope="prototype"/>**

**或者**

**<bean id="account" class="com.foo.DefaultAccount" singleton="false"/>**

**Request**

当一个bean的作用域为Request，表示在**一次HTTP请求中，一个bean定义对应一个实例**；即每个HTTP请求都会有各自的bean实例，它们依据某个bean定义创建而成。该作用域仅在基于web的Spring ApplicationContext情形下有效。考虑下面bean定义：

**<bean id="loginAction" class=cn.csdn.LoginAction" scope="request"/>**

针对每次HTTP请求，Spring容器会根据loginAction bean的定义创建一个全新的LoginAction bean实例，且该loginAction bean实例仅在当前HTTP request内有效，因此可以根据需要放心的更改所建实例的内部状态，而其他请求中根据loginAction bean定义创建的实例，将不会看到这些特定于某个请求的状态变化。**当处理请求结束，request作用域的bean实例将被销毁**。

**Session**

当一个bean的作用域为Session，表示在**一个HTTP Session中，一个bean定义对应一个实例**。该作用域仅在基于web的Spring ApplicationContext情形下有效。考虑下面bean定义：

**<bean id="userPreferences" class="com.foo.UserPreferences" scope="session"/>**

针对某个HTTP Session，Spring容器会根据userPreferences bean定义创建一个全新的userPreferences bean实例，且该userPreferences bean仅在当前HTTP Session内有效。与request作用域一样，可以根据需要放心的更改所创建实例的内部状态，而别的HTTP Session中根据userPreferences创建的实例，将不会看到这些特定于某个HTTP Session的状态变化。**当HTTP Session最终被废弃的时候，在该HTTP Session作用域内的bean也会被废弃掉**。

**Bean的自动装配**

* 自动装配是使用spring满足bean依赖的一种方法
* spring会在应用上下文中为某个bean寻找其依赖的bean。

Spring中bean有三种装配机制，分别是：

1. 在xml中显式配置；
2. 在java中显式配置；
3. 隐式的bean发现机制和自动装配。

这里我们主要讲第三种：**自动化的装配bean**。

Spring的自动装配需要从两个角度来实现，或者说是两个操作：

1. 组件扫描(component scanning)：spring会自动发现应用上下文中所创建的bean；
2. 自动装配(autowiring)：spring自动满足bean之间的依赖，也就是我们说的IoC/DI；

组件扫描和自动装配组合发挥巨大威力，使得显示的配置降低到最少。

**推荐不使用自动装配xml配置 , 而使用注解 .**

**7.1.byName**

**autowire byName (按名称自动装配)**

由于在手动配置xml过程中，常常发生字母缺漏和大小写等错误，而无法对其进行检查，使得开发效率降低。

采用自动装配将避免这些错误，并且使配置简单化。

1.修改bean配置，增加一个属性 autowire=“byName”

<bean id="user" class="com.kuang.pojo.User" autowire="byName">

<property name="str" value="qinjiang"/>

</bean>

2、再次测试，结果依旧成功输出！

3、我们将 cat 的bean id修改为 catXXX

4、再次测试， 执行时报空指针java.lang.NullPointerException。因为按byName规则找不对应set方法，真正的setCat就没执行，对象就没有初始化，所以调用时就会报空指针错误。

**小结：**

当一个bean节点带有 autowire byName的属性时。

1. 将查找其类中所有的set方法名，例如setCat，获得将set去掉并且首字母小写的字符串，即cat。
2. 去spring容器中寻找是否有此字符串名称id的对象。
3. 如果有，就取出注入；如果没有，就报空指针异常。

**7.2.byType**

**autowire byType (按类型自动装配)**

使用autowire byType首先需要保证：**同一类型的对象，在spring容器中唯一。如果不唯一，会报不唯一的异常。**

NoUniqueBeanDefinitionException

测试：

1、将user的bean配置修改一下 ： autowire=“byType”

2、测试，正常输出

3、在注册一个cat 的bean对象！

<bean id="dog" class="com.kuang.pojo.Dog"/>

<bean id="cat" class="com.kuang.pojo.Cat"/>

<bean id="cat2" class="com.kuang.pojo.Cat"/>

<bean id="user" class="com.kuang.pojo.User" autowire="byType">

<property name="str" value="qinjiang"/>

</bean>

4、测试，报错：NoUniqueBeanDefinitionException

5、删掉cat2，将cat的bean名称改掉！测试！因为是按类型装配，所以并不会报异常，也不影响最后的结果。甚至将id属性去掉，也不影响结果。

**7.3.使用注解**

jdk1.5开始支持注解，spring2.5开始全面支持注解。

准备工作：利用注解的方式注入属性。

1、在spring配置文件中引入context文件头

xmlns:context="http://www.springframework.org/schema/context"

http://www.springframework.org/schema/context

<http://www.springframework.org/schema/context/spring-context.xsd>

2、开启属性注解支持！

**<context:annotation-config/>**

#### @Autowired

* @Autowired是按**类型自动转配**的，不支持id匹配。
* 需要导入 spring-aop的包！

@Autowired(required=false) 说明：false，对象可以为null；true，对象必须存对象，不能为null。

#### @Qualifier

* @Autowired是根据类型自动装配的，加上@Qualifier则可以根据byName的方式自动装配
* @Qualifier**不能单独使用**。

#### @Resource

* @Resource如有指定的name属性，**先按该属性进行byName方式查找装配**；
* 其次再进行默认的byName方式进行装配；
* **如果以上都不成功，则按byType的方式自动装配**。
* 都不成功，则报异常。

**小结**

@Autowired与@Resource异同：

1、@Autowired与@Resource都可以用来装配bean。都可以写在字段上，或写在setter方法上。

2、@Autowired默认按类型装配（属于spring规范），默认情况下必须要求依赖对象必须存在，如果要允许null 值，可以设置它的required属性为false，如：@Autowired(required=false) ，如果我们想使用名称装配可以结合@Qualifier注解进行使用

3、@Resource（属于J2EE复返），默认按照名称进行装配，名称可以通过name属性进行指定。如果没有指定name属性，当注解写在字段上时，默认取字段名进行按照名称查找，如果注解写在setter方法上默认取属性名进行装配。当找不到与名称匹配的bean时才按照类型进行装配。但是需要注意的是，如果name属性一旦指定，就只会按照名称进行装配。

它们的作用相同都是用注解方式注入对象，但执行顺序不同。@Autowired先byType，@Resource先byName。’

#### **@Component**

**使用注解注入属性（替代了在配置文件当中配置步骤而已）**

为了更好的进行分层，Spring可以使用其它三个注解，功能一样，目前使用哪一个功能都一样。

* @Controller：controller层
* @Service：service层
* @Repository：dao层

写上这些注解，就相当于将这个类交给Spring管理装配了！

**8.使用注解开发**

[(192条消息) Spring-全面详解（学习总结）\_spring策划本天成\_策谋本天成的博客-CSDN博客](https://blog.csdn.net/weixin_44207403/article/details/106736102?ops_request_misc=%257B%2522request%255Fid%2522%253A%2522167884016116800211590196%2522%252C%2522scm%2522%253A%252220140713.130102334..%2522%257D&request_id=167884016116800211590196&biz_id=0&utm_medium=distribute.pc_search_result.none-task-blog-2~all~top_positive~default-1-106736102-null-null.142%5ev73%5econtrol,201%5ev4%5eadd_ask,239%5ev2%5einsert_chatgpt&utm_term=spring&spm=1018.2226.3001.4187)

**8.1.Bean的实现**

**1、配置扫描哪些包下的注解**

<!--指定注解扫描包-->

<context:component-scan base-package="com.kuang.pojo"/>

2、在指定包下编写类，增加注解

@Component("user")

// 相当于配置文件中 <bean id="user" class="当前注解的类"/>

public class User {

public String name = "秦疆";

}

3、测试

@Test

public void test(){

ApplicationContext applicationContext =

new ClassPathXmlApplicationContext("beans.xml");

User user = (User) applicationContext.getBean("user");

System.out.println(user.name);

}

**8.2.属性注入**

**使用注解注入属性**

可以不用提供set方法，直接在直接名上添加@value(“值”)

@Component("user")

// 相当于配置文件中 <bean id="user" class="当前注解的类"/>

public class User {

@Value("秦疆")

// 相当于配置文件中 <property name="name" value="秦疆"/>

public String name;

}

**如果提供了set方法，在set方法上添加@value(“值”);**

@Component("user")

public class User {

public String name;

**@Value("秦疆")**

public void **setName**(String name) {

this.name = name;

}

}

**8.3.衍生注解**

我们这些注解，就是替代了在配置文件当中配置步骤而已！更加的方便快捷！

**@Component三个衍生注解**

为了更好的进行分层，Spring可以使用其它三个注解，功能一样，目前使用哪一个功能都一样。

* @Controller：controller层
* @Service：service层
* @Repository：dao层

写上这些注解，就相当于将这个类交给Spring管理装配了！

**8.4.自动装配注解**

在Bean的自动装配已经讲过了，可以回顾！

**8.5.作用域**

@scope

* **singleton**：默认的，Spring会采用**单例模式**创建这个对象。关闭工厂 ，所有的对象都会销毁。
* **prototype**：**多例模式**。关闭工厂 ，所有的对象不会销毁。内部的垃圾回收机制会回收

@Controller("user")

**@Scope("prototype")**

public class User {

@Value("秦疆")

public String name;

}

**8.6.小结**

**XML与注解比较**

* XML可以适用任何场景 ，结构清晰，维护方便
* 注解不是自己提供的类使用不了，开发简单方便

**xml与注解整合开发** ：推荐最佳实践

* xml管理Bean
* 注解完成属性注入
* 使用过程中， 可以不用扫描，扫描是为了类上的注解

<context:annotation-config/>

* 1

作用：

* 进行注解驱动注册，从而使注解生效
* 用于激活那些已经在spring容器里注册过的bean上面的注解，也就是显示的向Spring注册
* 如果不扫描包，就需要手动配置bean
* 如果不加注解驱动，则注入的值为null！